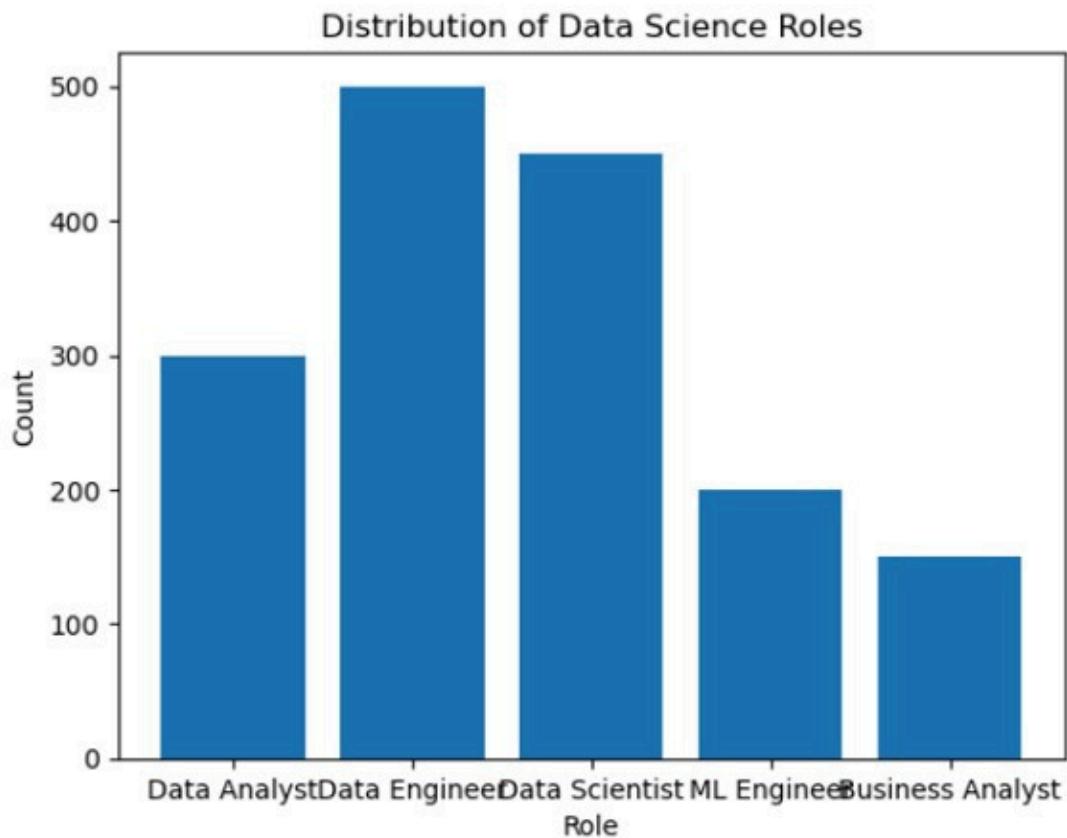


Analyze and visualize the distribution of various data science roles from a dataset

```
roles= ['Data Analyst', 'Data Engineer', 'Data Scientist', 'ML Engineer',  
'Business Analyst']  
  
counts =[300, 500, 450, 200, 150]  
  
plt.bar(roles, counts)  
  
plt.title('Distribution of Data Science Roles')  
  
plt.xlabel('Role')  
  
plt.ylabel('Count')  
  
plt.show()
```



CONDUCT AN EXPERIMENT TO ENCRYPT AND DECRYPT GIVEN SENSITIVE DATA.

```
from cryptography.fernet import Fernet
key=Fernet.generate_key()
f=Fernet(key)
token=f.encrypt(b"My name is Dinisha R")
token
b'...'
f.decrypt(token)
b'My name is Dinisha'
key=Fernet.generate_key()
cipher_suite=Fernet(key)
plain_text=b'My name is Dinisha'
cipher_text=cipher_suite.encrypt(plain_text)
decrypt_text=cipher_suite.decrypt(cipher_text)
print("Original Data",plain_text)
print("Encrypted Data",cipher_text)
print("Decrypted Data",decrypt_text)
```

Original Data b'My name is

Encrypted Data b'gAAAAABmvrGenCor03j3aGQZW-H0fVnRCA9RQbad5C_jow_zvapDqh71XH-iUq2sRgf1Mpu8PyAx162uK6RdLmBFQhvU3iA6SgPLgv6B9VsH-XIgPJN8wQ='

Decrypted Data b'My name is

Count the frequency of occurrence of a word in a body of text is often needed during text processing.

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import gutenberg
nltk.download('gutenberg')
nltk.download('punkt')
sample = gutenberg.raw("austen-emma.txt")
token = word_tokenize(sample)
wlist = []
for i in range(50):
    wlist.append(token[i])
wordfreq = [wlist.count(w) for w in wlist]
print("Pairs\n" + str(list(zip(wlist, wordfreq))))
```

```
[nltk_data] Downloading package gutenberg to
[nltk_data]     C:\Users\DELL\AppData\Roaming\nltk_data...
[nltk_data] Package gutenberg is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\DELL\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
```

Pairs

```
[('[' , 1), ('Emma' , 2), ('by' , 1), ('Jane' , 1), ('Austen' , 1), ('1816' , 1), (']' , 1), ('VOLUME' , 1), ('I' , 2), ('CHAPTER' , 1),
('I' , 2), ('Emma' , 2), ('Woodhouse' , 1), (',' , 5), ('handsome' , 1), (',' , 5), ('clever' , 1), (',' , 5), ('and' , 3), ('rich' , 1),
(',' , 5), ('with' , 2), ('a' , 1), ('comfortable' , 1), ('home' , 1), ('and' , 3), ('happy' , 1), ('disposition' , 1), (',' , 5), ('see
med' , 1), ('to' , 1), ('unite' , 1), ('some' , 1), ('of' , 2), ('the' , 2), ('best' , 1), ('blessings' , 1), ('of' , 2), ('existence' ,
1), (';' , 1), ('and' , 3), ('had' , 1), ('lived' , 1), ('nearly' , 1), ('twenty-one' , 1), ('years' , 1), ('in' , 1), ('the' , 2), ('wo
rld' , 1), ('with' , 2)]
```

Data Cleaning

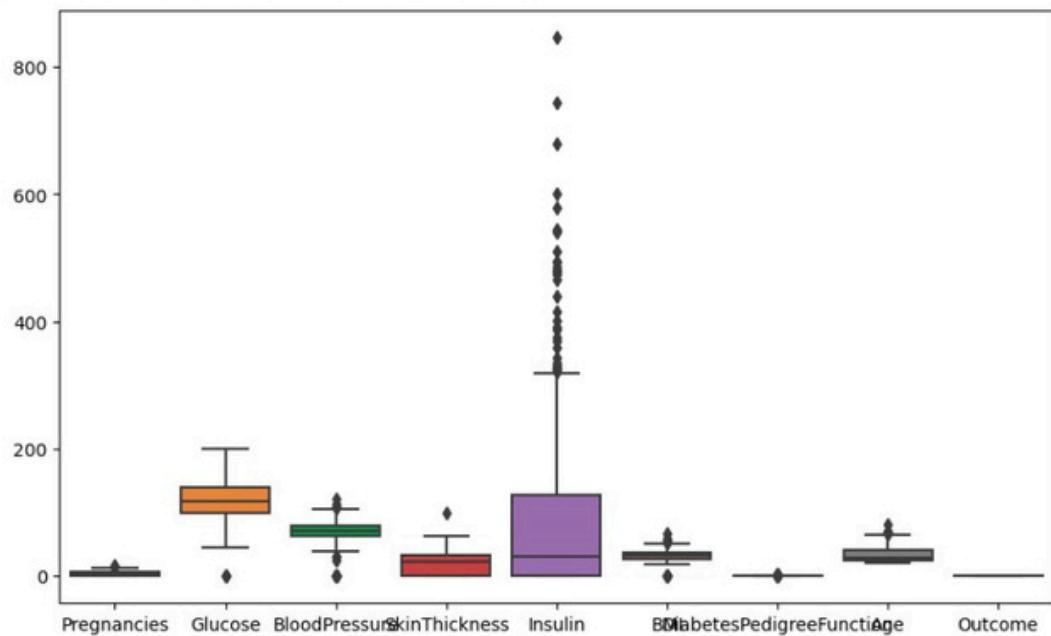
```
print(db.isnull())
db.fillna(db.mean(), inplace=True)
print(db.isnull().sum())
```

```
Pregnancies Glucose BloodPressure SkinThickness Insulin BMI \
0 False False False False False False
1 False False False False False False
2 False False False False False False
3 False False False False False False
4 False False False False False False
.. ...
763 False False False False False False
764 False False False False False False
765 False False False False False False
766 False False False False False False
767 False False False False False False

DiabetesPedigreeFunction Age Outcome
0 False False False
1 False False False
2 False False False
3 False False False
4 False False False
.. ...
763 False False False
764 False False False
765 False False False
766 False False False
767 False False False

[768 rows x 9 columns]
Pregnancies      0
Glucose          0
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
DiabetesPedigreeFunction 0
Age              0
Outcome          0
dtype: int64
```

```
import numpy as np
plt.figure(figsize=(10,6))
sns.boxplot(data=db)
plt.show()
from scipy import stats
diabetes_df=db[(np.abs(stats.zscore(db))<3).all(axis=1)]
```



Data Collection and Initial Exploration

```
import pandas as pd
import matplotlib.pyplot as plt
db = pd.read_csv("diabetes.csv")
print(db.head())
```

```
Pregnancies Glucose BloodPressure SkinThickness Insulin BMI \
0           6      148          72        35     0  33.6
1           1       85          66        29     0  26.6
2           8      183          64        0     0  23.3
3           1       89          66        23    94  28.1
4           0      137          40        35   168  43.1
```

```
DiabetesPedigreeFunction Age Outcome
0            0.627   50      1
1            0.351   31      0
2            0.672   32      1
3            0.167   21      0
4            2.288   33      1
```

```
print(db.info)
print(db.describe)
import seaborn as sns
db.hist(bins=50,figsize=(20,15))
plt.show()
sns.pairplot(db)
plt.show()
```

```
<bound method DataFrame.info of      Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI \
0           6      148          72        35     0  33.6
1           1       85          66        29     0  26.6
2           8      183          64        0     0  23.3
3           1       89          66        23    94  28.1
4           0      137          40        35   168  43.1
...
...           ...
763          10     101          76        48   180  32.9
764          2      122          70        27     0  36.8
765          5      121          72        23   112  26.2
766          1      126          60        0     0  30.1
767          1       93          70        31     0  30.4
```

```
DiabetesPedigreeFunction Age Outcome
0            0.627   50      1
1            0.351   31      0
2            0.672   32      1
3            0.167   21      0
4            2.288   33      1
...
...           ...
763          0.171   63      0
764          0.340   27      0
765          0.245   30      0
766          0.349   47      1
767          0.315   23      0
```

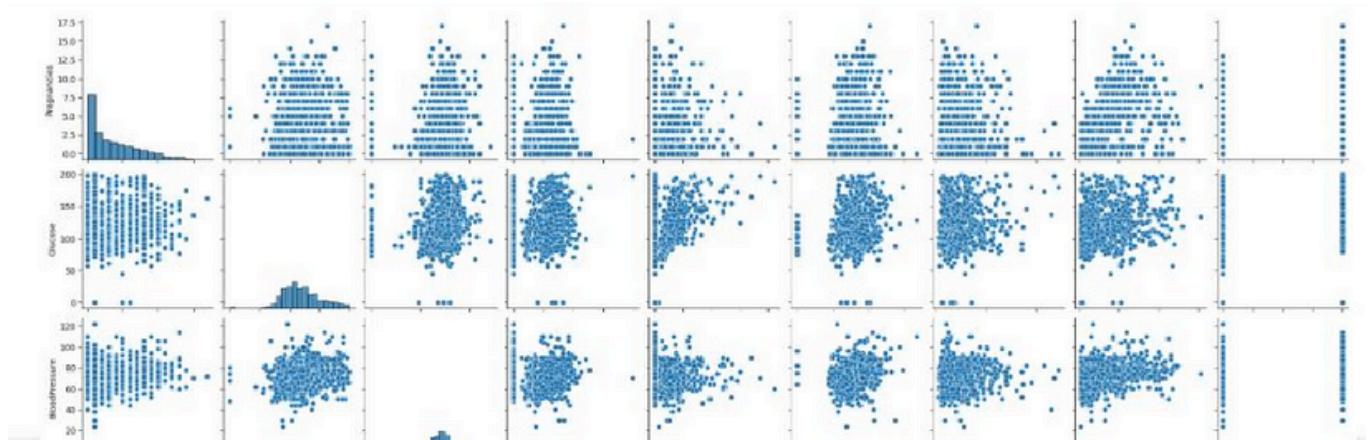
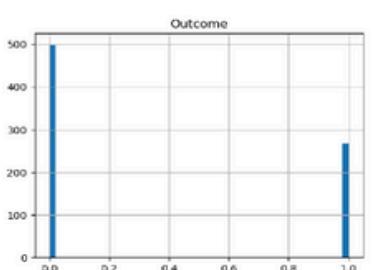
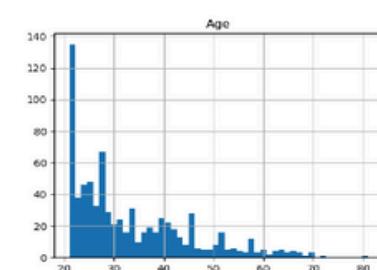
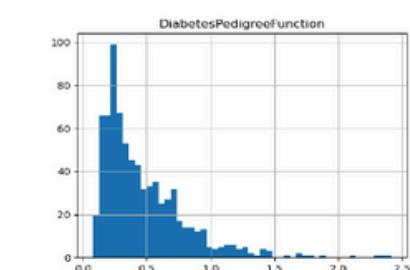
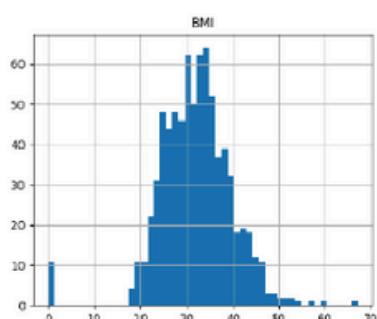
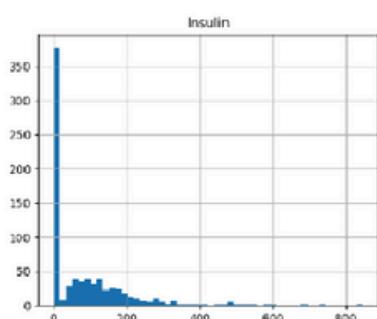
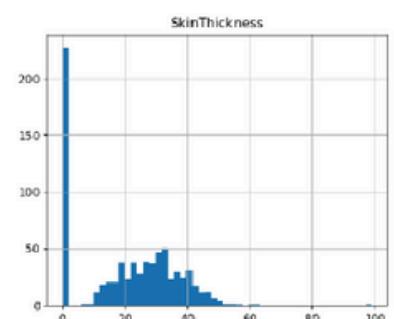
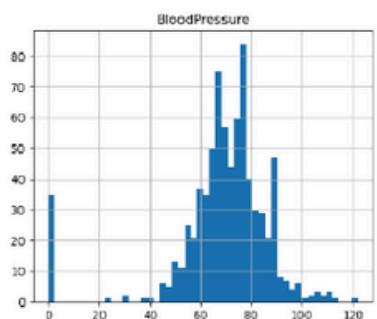
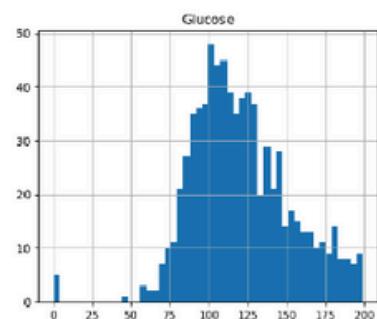
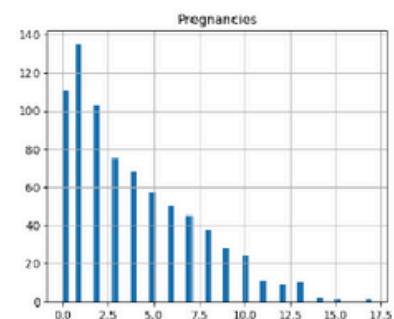
```
[768 rows x 9 columns]>
<bound method NDFrame.describe of      Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI \
0           6      148          72        35     0  33.6
1           1       85          66        29     0  26.6
2           8      183          64        0     0  23.3
3           1       89          66        23    94  28.1
4           0      137          40        35   168  43.1
...
...           ...
763          10     101          76        48   180  32.9
764          2      122          70        27     0  36.8
765          5      121          72        23   112  26.2
766          1      126          60        0     0  30.1
767          1       93          70        31     0  30.4
```

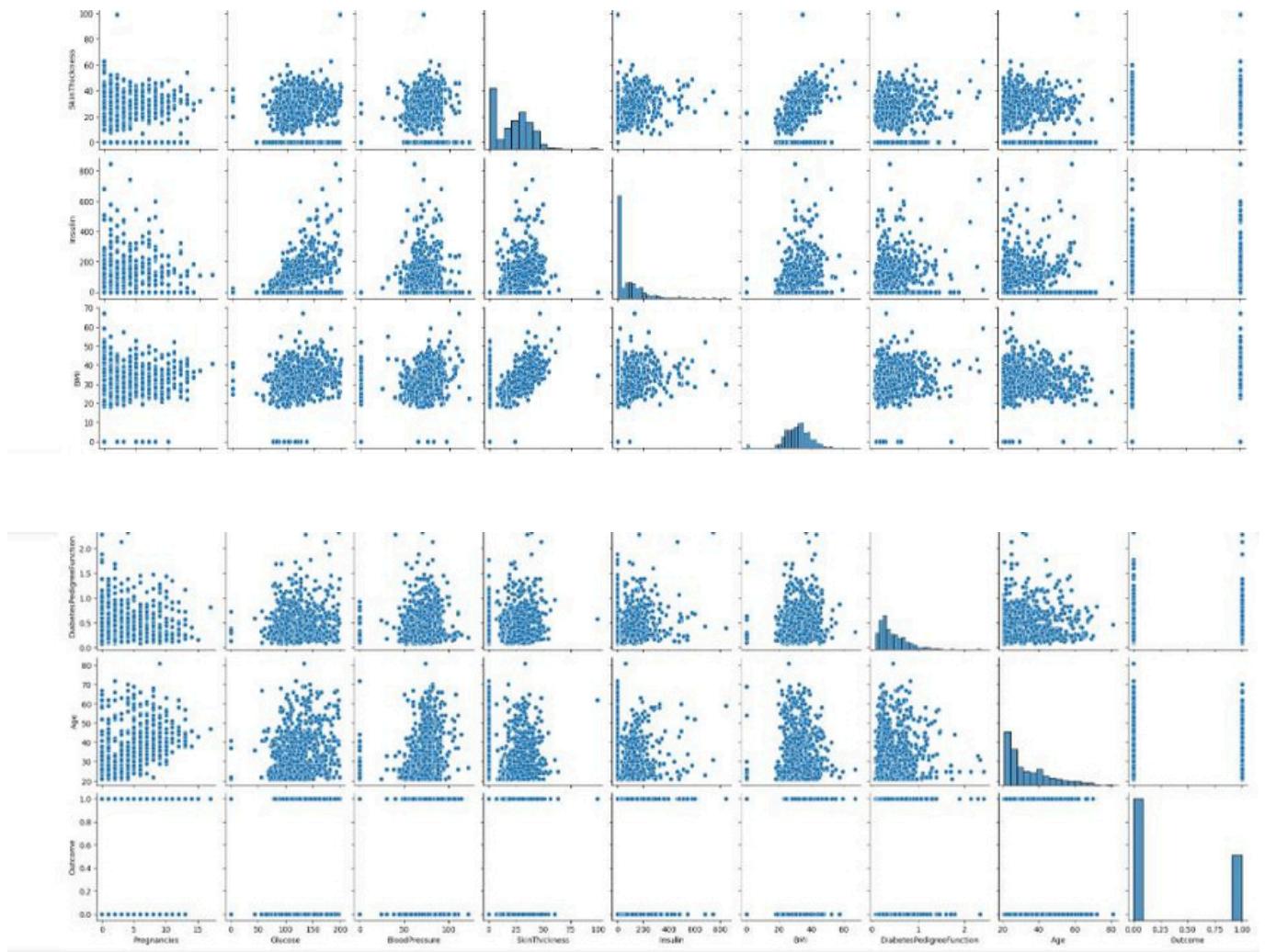
```

DiabetesPedigreeFunction  Age  Outcome
0                         0.627  50    1
1                         0.351  31    0
2                         0.672  32    1
3                         0.167  21    0
4                         2.288  33    1
..
763                        ...   ...
764                        ...   ...
765                        ...   ...
766                        ...   ...
767                        ...   ...

```

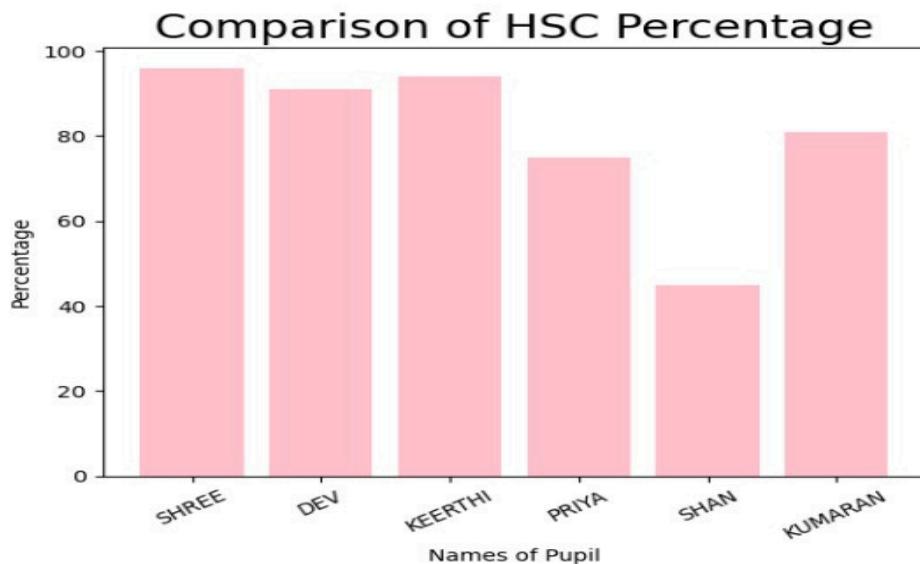
[768 rows x 9 columns]>





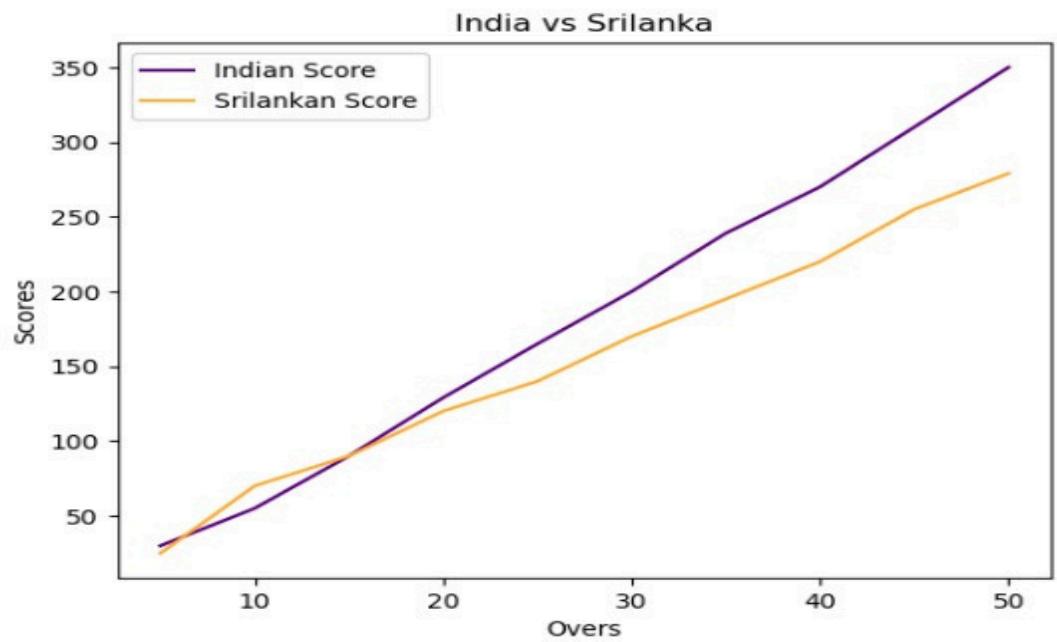
Experiment to show data visualization using bar chart

```
In [1]: import matplotlib.pyplot as hscmark
import numpy as np
Names = ['SHREE', 'DEV', 'KEERTHI', 'PRIYA', 'SHAN', 'KUMARAN']
xaxis = np.arange(len(Names))
Percentage_hsc = [96, 91, 94, 75, 45, 81]
hscmark.bar(Names, Percentage_hsc,color='pink')
hscmark.xticks(xaxis, Names, rotation=30)
hscmark.xlabel('Names of Pupil')
hscmark.ylabel('Percentage')
hscmark.title('Comparison of HSC Percentage', fontsize=20, color='black')
hscmark.show()
```



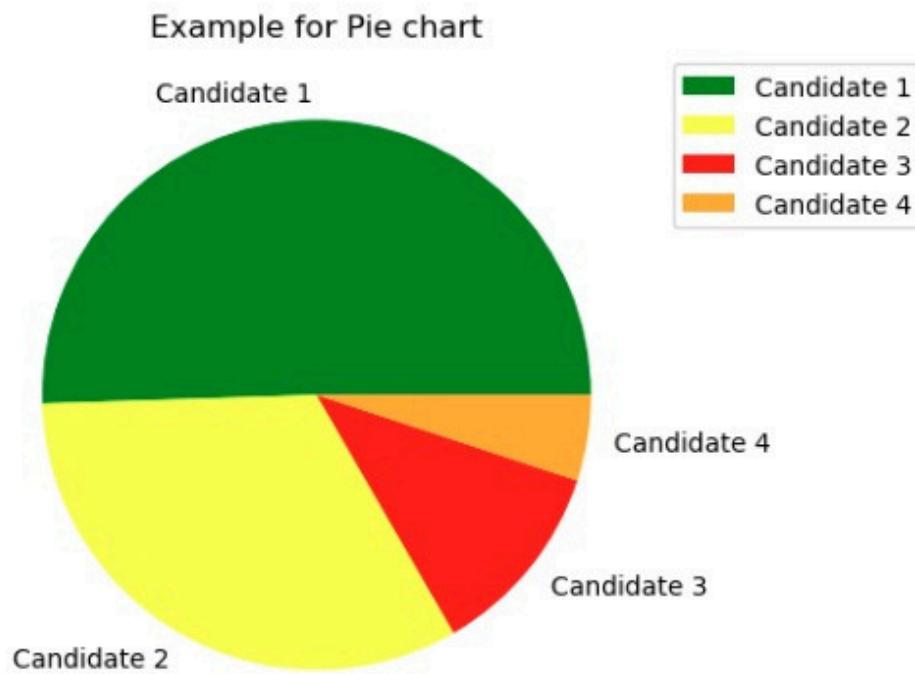
Experiment to show data visualization using line plot

```
In [1]: import matplotlib.pyplot as cricket
Overs=list(range(5,51,5))
Indian_Score=[30,55,90,129,165,200,239,270,310,350]
Srilankan_Score=[25,70,90,120,140,170,195,220,255,279]
cricket.plot(Overs,Indian_Score,label='Indian Score',color='indigo')
cricket.plot(Overs,Srilankan_Score,label='Srilankan Score',color='orange')
cricket.title("India vs Srilanka")
cricket.xlabel("Overs")
cricket.ylabel("Scores")
cricket.legend()
cricket.show()
```



Experiment to show data visualization using pie chart

```
In [1]: import numpy as np
import matplotlib.pyplot as election
roles=['Candidate 1','Candidate 2','Candidate 3','Candidate 4']
count=np.array([100,65,23,10])
colours = ['green','yellow','red','orange']
election.pie(count,labels=roles,colors=colours)
election.legend(loc="upper left",bbox_to_anchor=(1,1))
election.title("Example for Pie chart")
election.show()
```



Experiments on Structured, Unstructured and Semi Structured

```
import pandas as pd
structured_data=pd.DataFrame({ 'ID': [1,2,3], 'Name': ['Alice', 'Bob', 'Charlie'],
'Age': [25,30,35] })
print("Structured data: \n", structured_data)
unstructured_data="This is an example of unstructured data. It can be a piece of text, an image, or a video file."
print("Unstructured data: \n", unstructured_data)

semi_structured={'ID': 1, 'Name': 'Alice', 'Attributes': {'Height':165, 'Weight':68}}
print("Semi Structed data: \n", semi_structured)
```

output:

```
Structured data :
   ID      Name  Age
0   1      Alice  25
1   2        Bob  30
2   3    Charlie  35

Unstructured data :
This is an example of unstructured data. It can be a piece of text, an image, or a video file.

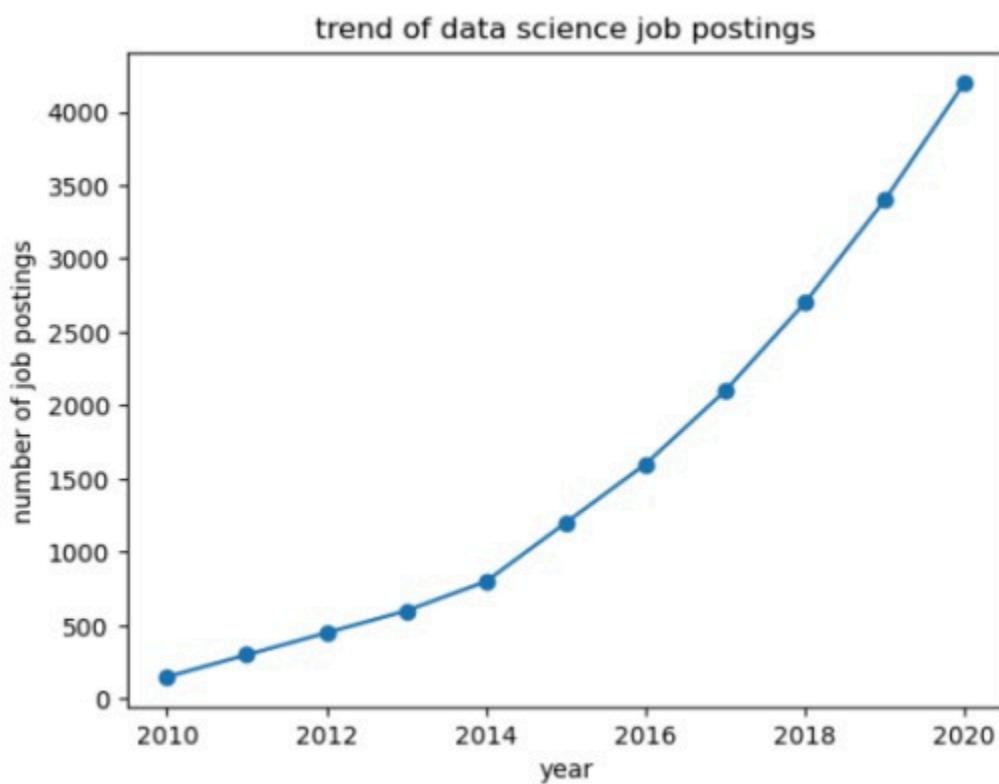
Semi Structed data:
{'ID': 1, 'Name': 'Alice', 'Attributes': {'Height': 165, 'Weight': 68}}
```

Using Pandas for data manipulation and Matplotlib for visualization

```
Import pandas as pd
```

```
Import matplotlib.pyplot as plt
```

```
data ={'Year': list(range (2010,2021)), 'job posting': [150, 300, 450, 600, 800, 1200, 1600, 2100, 2700,  
3400,4200]}  
df = pd.DataFrame(data)  
  
plt.plot (df['Year'], df['job posting'], marker='o')  
  
plt.title('trend of data science job postings')  
  
plt.xlabel('year')  
  
plt.ylabel('number of job postings')  
  
plt.show()
```



4: DATA PREPROCESSING

NAME: HARSHA VARDHANAN V

ROLL NO: 230701108

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler

df = pd.read_csv('Hotel_Dataset.csv')

print("Original Dataset:")
print(df.head())

df.replace({'Bill': { -1: np.nan, -99999: np.nan, 0: np.nan},
             'NoOfPax': {-1: np.nan, 0: np.nan},
             'EstimatedSalary': {-99999: np.nan, 0: np.nan},
             'Rating(1-5)': { -1: np.nan}}, inplace=True)

df = df.drop_duplicates()

df['Bill'] = df['Bill'].fillna(df['Bill'].mean())
df['NoOfPax'] = df['NoOfPax'].fillna(df['NoOfPax'].mode()[0]) # Mode for categorical-like column
df['EstimatedSalary'] = df['EstimatedSalary'].fillna(df['EstimatedSalary'].mean())
df['Rating(1-5)'] = df['Rating(1-5)'].fillna(df['Rating(1-5)'].mode()[0])

label_encoder = LabelEncoder()
df['Hotel'] = label_encoder.fit_transform(df['Hotel'])
df['FoodPreference'] = label_encoder.fit_transform(df['FoodPreference'])

df = pd.get_dummies(df, columns=['Age_Group'], drop_first=True)
```

```

scaler = StandardScaler()
df[['Bill', 'EstimatedSalary']] = scaler.fit_transform(df[['Bill', 'EstimatedSalary']])

print("\nPreprocessed Dataset:")
print(df.head())

df.to_csv('Preprocessed_Hotel_Dataset.csv', index=False)

```

Original Dataset:								
	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	\
0	1	20-25	4	Ibis	veg	1300	2	
1	2	30-35	5	LemonTree	Non-Veg	2000	3	
2	3	25-30	6	RedFox	Veg	1322	2	
3	4	20-25	-1	LemonTree	Veg	1234	2	
4	5	35+	3	Ibis	Vegetarian	989	2	

EstimatedSalary Age_Group.1		
	EstimatedSalary	Age_Group.1
0	40000	20-25
1	59000	30-35
2	30000	25-30
3	120000	20-25
4	45000	35+

EstimatedSalary Age_Group.1		
	EstimatedSalary	Age_Group.1
0	40000	20-25
1	59000	30-35
2	30000	25-30
3	120000	20-25
4	45000	35+

Preprocessed Dataset:							
	CustomerID	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	\
0	1	4.0	0	4	0.131957	2.0	
1	2	5.0	2	0	0.392446	3.0	
2	3	6.0	3	1	0.140143	2.0	
3	4	3.0	2	1	0.107396	2.0	
4	5	3.0	0	2	0.016225	2.0	

Preprocessed Dataset:

	CustomerID	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	\
0	1	4.0	0		4 0.131957	2.0	
1	2	5.0	2		0 0.392446	3.0	
2	3	6.0	3		1 0.140143	2.0	
3	4	3.0	2		1 0.107396	2.0	
4	5	3.0	0		2 0.016225	2.0	

	EstimatedSalary	Age_Group.1	Age_Group_25-30	Age_Group_30-35	\
0	-0.631656	20-25	False	False	
1	-0.420194	30-35	False	True	
2	-0.742952	25-30	True	False	
3	0.258711	20-25	False	False	
4	-0.576008	35+	False	False	

	Age_Group_35+
0	False
1	False
2	False
3	False
4	True

5: EDA quantitative and qualitative plot

NAME: HARSHA VARDHANAN V
ROLL NO :230701108

```
import pandas as pd import
    matplotlib.pyplot as plt
    import seaborn as sns
data = {
    'total_bill': [16.99, 10.34, 21.01, 23.68, 24.59],
    'tip': [1.01, 1.66, 3.50, 3.31, 3.61],
    'sex': ['Female', 'Male', 'Male', 'Male', 'Female'],
    'smoker': ['No', 'No', 'No', 'No', 'No'],
    'day': ['Sun', 'Sun', 'Sun', 'Sun', 'Sun'],
    'time': ['Dinner', 'Dinner', 'Dinner', 'Dinner', 'Dinner'],
    'size': [2, 3, 3, 2, 4]
}

df = pd.DataFrame(data)

# Set up Seaborn style for plots
sns.set(style="whitegrid")

# -----
# Quantitative Plots
# -----



plt.figure(figsize=(8, 6))
sns.histplot(df['total_bill'], kde=True, color='blue', bins=10)
plt.title('Distribution of Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Frequency')
plt.show()



plt.figure(figsize=(8, 6))
sns.histplot(df['tip'], kde=True, color='green', bins=10)
plt.title('Distribution of Tip')
```

```
plt.xlabel('Tip')
plt.ylabel('Frequency')
plt.show()
```

```
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['total_bill'], color='orange')
plt.title('Boxplot of Total Bill')
plt.xlabel('Total Bill')
plt.show()
```

```
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['total_bill'], y=df['tip'], color='purple')
plt.title('Total Bill vs Tip')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```

```
# -----
# Qualitative Plots
# -----
```

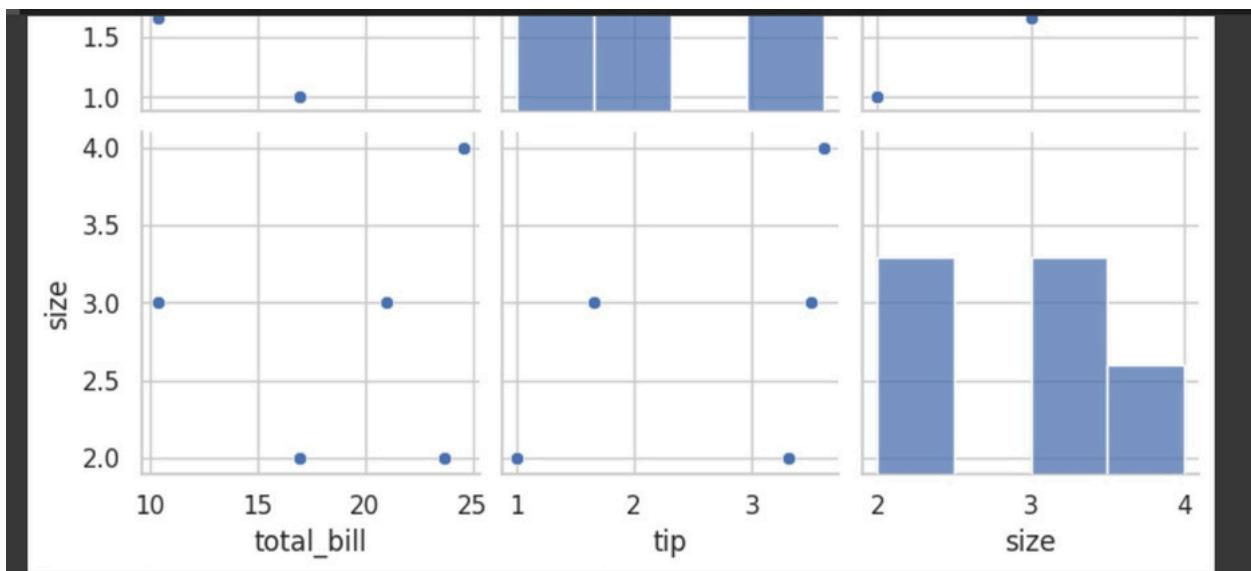
```
plt.figure(figsize=(8, 6))
sns.countplot(x='sex', data=df, palette='Set2')
plt.title('Count of Customers by Sex')
plt.xlabel('Sex')
plt.ylabel('Count')
plt.show()
plt.figure(figsize=(8, 6))
sns.countplot(x='smoker', data=df, palette='Set3')
plt.title('Count of Smokers vs Non-Smokers')
plt.xlabel('Smoker')
plt.ylabel('Count')
plt.show()
```

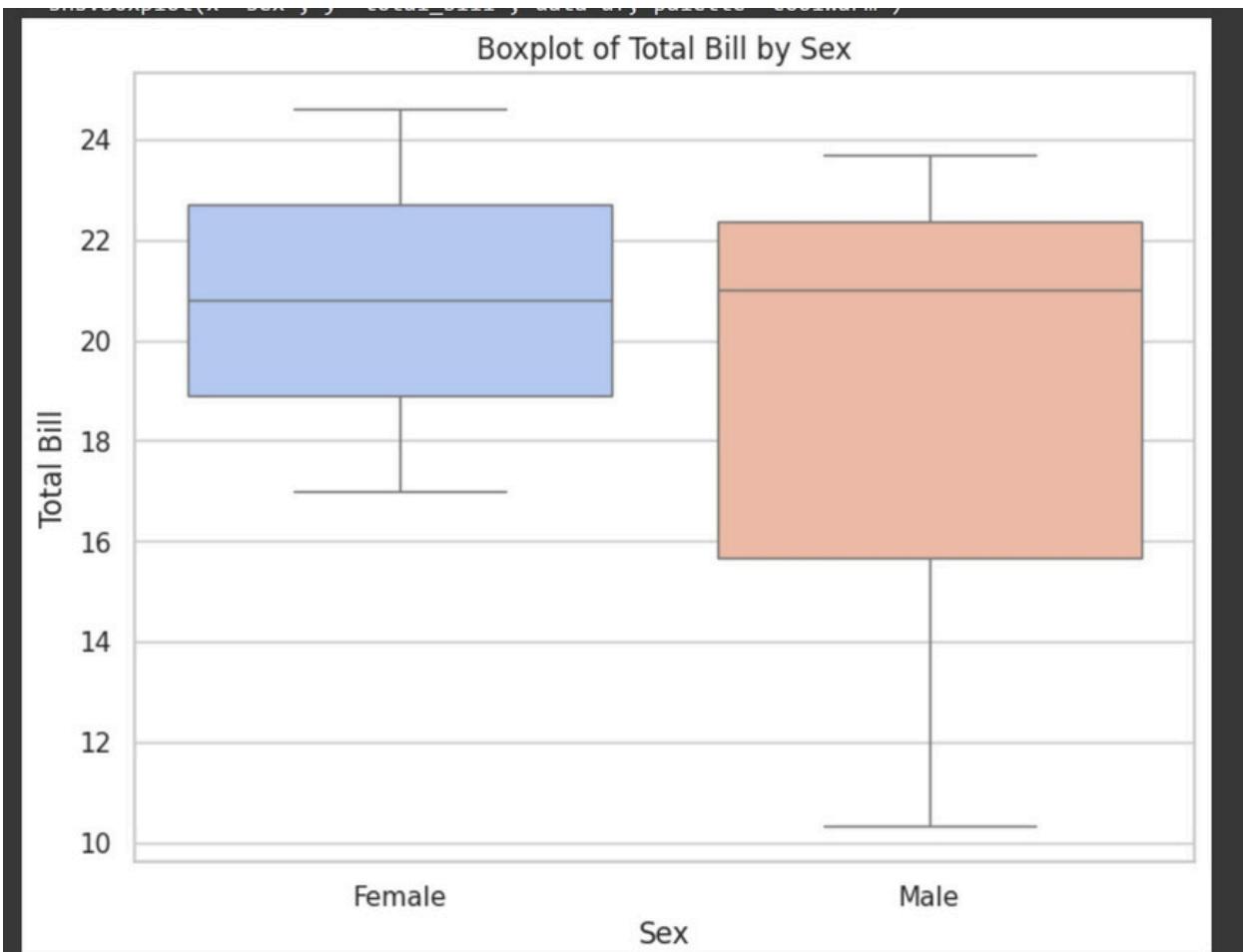
```
plt.figure(figsize=(8, 6))
sns.countplot(x='day', data=df, palette='muted')
plt.title('Count of Customers by Day')
plt.xlabel('Day')
plt.ylabel('Count')
plt.show()
```

```
plt.figure(figsize=(8, 6))
sns.countplot(x='time', data=df, palette='pastel')
plt.title('Count of Customers by Time')
plt.xlabel('Time')
plt.ylabel('Count')
plt.show()
```

```
sns.pairplot(df[['total_bill', 'tip', 'size']])
plt.suptitle('Pairplot: Total Bill, Tip, and Size', y=1.02)
plt.show()
```

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='sex', y='total_bill', data=df, palette='coolwarm')
plt.title('Boxplot of Total Bill by Sex')
plt.xlabel('Sex')
plt.ylabel('Total Bill')
plt.show()
```





6: RANDOM SAMPLING AND SAMPLING DISTRIBUTION

NAME: HARSHA VARDHANAN V
ROLL NO: 230701108

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

population = np.random.normal(loc=50, scale=10, size=10000) # Mean=50, SD=10,
Population size=10,000

plt.figure(figsize=(8, 6))
plt.hist(population, bins=50, color='skyblue', edgecolor='black', alpha=0.7)
plt.title('Population Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

sample_size = 100
random_sample = np.random.choice(population, size=sample_size, replace=False)

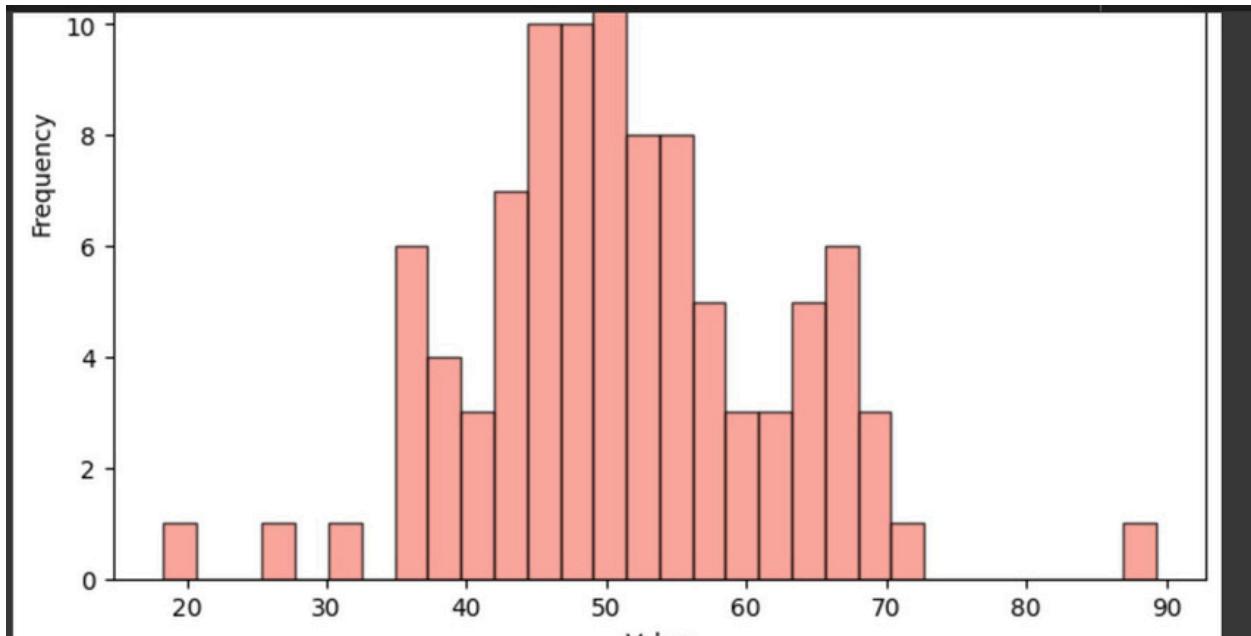
plt.figure(figsize=(8, 6))
plt.hist(random_sample, bins=30, color='salmon', edgecolor='black', alpha=0.7)
plt.title(f'Random Sample Distribution (Sample Size = {sample_size})')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

num_samples = 1000 # Number of samples to draw
sample_means = []

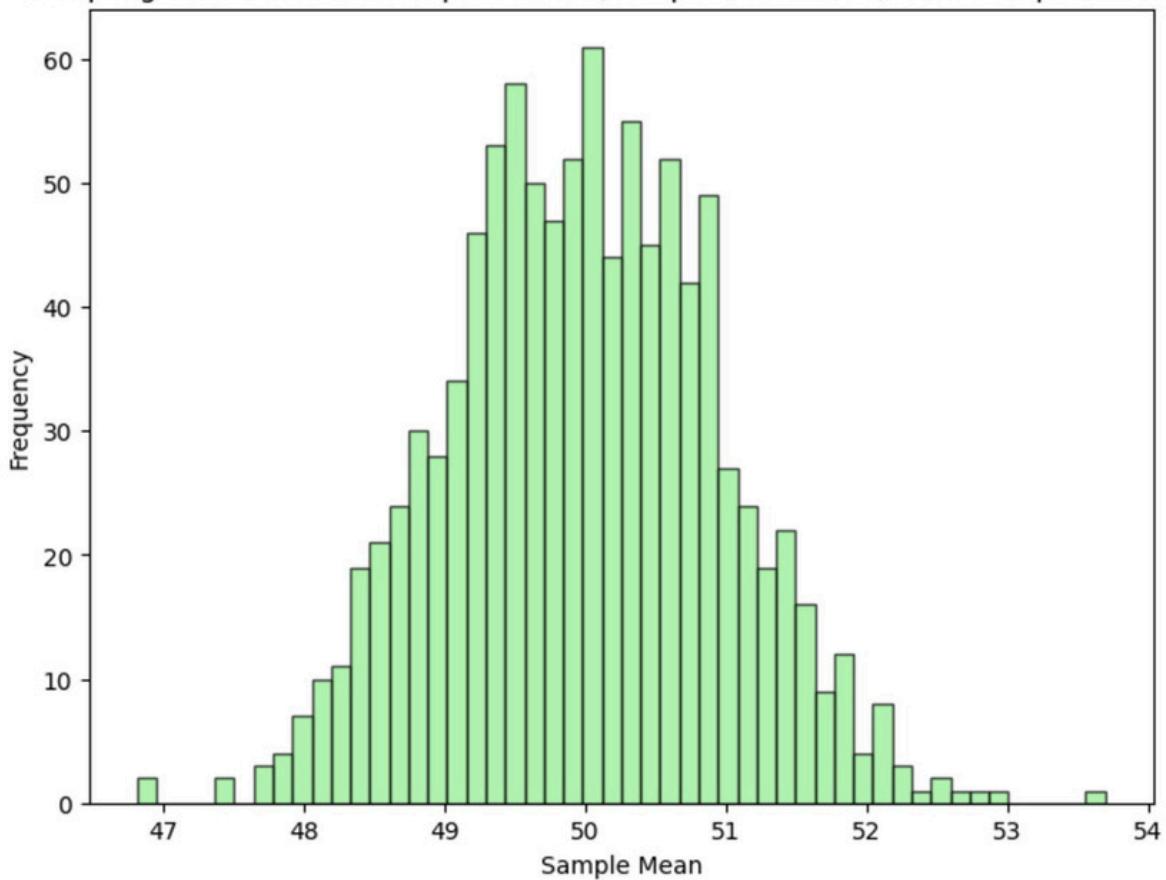
for _ in range(num_samples):
    sample = np.random.choice(population, size=sample_size, replace=False)
    sample_means.append(np.mean(sample))
```

```
plt.figure(figsize=(8, 6))
plt.hist(sample_means, bins=50, color='lightgreen', edgecolor='black', alpha=0.7)
plt.title(f'Sampling Distribution of Sample Means (Sample Size = {sample_size}, Num Samples = {num_samples})')
plt.xlabel('Sample Mean')
plt.ylabel('Frequency')
plt.show()
```

```
print(f"Mean of population: {np.mean(population)}")
print(f"Mean of sampling distribution: {np.mean(sample_means)}")
print(f"Standard Deviation of population: {np.std(population)}")
print(f"Standard Deviation of sampling distribution: {np.std(sample_means)}")
```



Sampling Distribution of Sample Means (Sample Size = 100, Num Samples = 1000)



7. Z-TEST

NAME : HARSHA VARDHANAN V

CLASS: CSE-B

ROLL NO : 230701108

CODE:

```
import numpy as np
import scipy.stats as stats
# Define the sample data (hypothetical weights in grams)
sample_data = np.array([152, 148, 151, 149, 147, 153, 150, 148, 152,
149, 151, 150, 149, 152, 151, 148, 150, 152, 149, 150, 148, 153, 151,
150, 149, 152, 148, 151, 150, 153])
# Population mean under the null hypothesis
population_mean = 150
# Calculate sample statistics
sample_mean = np.mean(sample_data)
sample_std = np.std(sample_data, ddof=1) # Using sample standard deviation
# Number of observations
n = len(sample_data)
# Calculate the Z-statistic
z_statistic = (sample_mean - population_mean) / (sample_std /
np.sqrt(n))
# Calculate the p-value
p_value = 2 * (1 - stats.norm.cdf(np.abs(z_statistic))) # Two-tailed test
```

```
# Print results

print(f"Sample Mean: {sample_mean:.2f}")

print(f"Z-Statistic: {z_statistic:.4f}")

print(f"P-Value: {p_value:.4f}")

# Decision based on the significance level

alpha = 0.05

if p_value < alpha:

    print("Reject the null hypothesis: The average weight is significantly different from 150
grams.")

else:

    print("Fail to reject the null hypothesis: There is no significant difference in average weight
from 150 grams.")
```

OUTPUT:

```
→ Sample Mean: 150.20
Z-Statistic: 0.6406
P-Value: 0.5218
Fail to reject the null hypothesis: There is no significant difference in average weight from 150 grams.
```

8. T-TEST

NAME : HARSHA VARDHANAN V

CLASS: CSE-B

ROLL NO : 230701108

```
CODE: import numpy as np import scipy.stats as stats  
# Set a random seed for reproducibility  
np.random.seed(42) # Generate hypothetical sample  
data (IQ scores) sample_size = 25 sample_data =  
np.random.normal(loc=102, scale=15,  
size=sample_size) # Mean IQ of 102, SD of 15 #  
Population mean under the null hypothesis  
population_mean = 100 # Calculate sample statistics  
sample_mean = np.mean(sample_data) sample_std =  
np.std(sample_data, ddof=1) n = len(sample_data) #  
Calculate the T-statistic and p-value t_statistic,  
p_value = stats.ttest_1samp(sample_data,
```

```
population_mean)

# Print results

print(f"Sample Mean: {sample_mean:.2f}")

print(f"T-Statistic: {t_statistic:.4f}")

print(f"P-Value: {p_value:.4f}")

# Decision based on the significance level

alpha = 0.05

if p_value < alpha:

    print("Reject the null hypothesis: The average IQ score is

significantly different from 100.")

else:

    print("Fail to reject the null hypothesis: There is no

significant difference in average IQ score from 100.")
```

OUTPUT:

```
✉ Sample Mean: 99.55
T-Statistic: -0.1577
P-Value: 0.8760
Fail to reject the null hypothesis: There is no significant difference in average IQ score from 100.
```

10.FEATURE SCALING

NAME : HARSHA VARDHANAN V

ROLL NO : 230701108

AIM: To do feature scaling in the given dataset.

```
import numpy as np
import pandas as pd
df=pd.read_csv('Data.csv')
df.head()
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes

```
df.Country.fillna(df.Country.mode()[0],inplace=True)
features=df.iloc[:, :-1].values
label=df.iloc[:, -1].values
from sklearn.impute import SimpleImputer
age=SimpleImputer(strategy="mean",missing_values=np.nan)
Salary=SimpleImputer(strategy="mean",missing_values=np.nan)
age.fit(features[:, [1]])
```

`df.Country.fillna(df.Country.mode()[0],inplace=True)`

FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment
The behavior will change in pandas 3.0. This `inplace` method will never work because the intermediate object on which we are setting values

For example, when doing `'df[col].method(value, inplace=True)'`, try using `'df.method({col: value}, inplace=True)'` or `df[col] = df[col].meth`

```
df.Country.fillna(df.Country.mode()[0],inplace=True)
```

SimpleImputer  

```
SimpleImputer()
```

```
Salary.fit(features[:,[2]])
```

```
Salary.fit(features[:,[2]])
```

SimpleImputer()

```
SimpleImputer()
```

```
SimpleImputer()
```

SimpleImputer()

```
features[:,[1]]=age.transform(features[:,[1]])
```

```
features[:,[2]]=Salary.transform(features[:,[2]])
```

```
features
```

```
array([[ 'France', 44.0, 72000.0],
       [ 'Spain', 27.0, 48000.0],
       [ 'Germany', 30.0, 54000.0],
       [ 'Spain', 38.0, 61000.0],
       [ 'Germany', 40.0, 63777.7777777778],
       [ 'France', 35.0, 58000.0],
       [ 'Spain', 38.7777777777778, 52000.0],
       [ 'France', 48.0, 79000.0],
       [ 'Germany', 50.0, 83000.0],
       [ 'France', 37.0, 67000.0]], dtype=object)
```

```
from sklearn.preprocessing import OneHotEncoder
```

```
oh = OneHotEncoder(sparse_output=False)
```

```
Country=oh.fit_transform(features[:,[0]])
```

```
Country
```

```
array([[1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.]])
```

```
final_set=np.concatenate((Country,features[:,[1,2]]),axis=1)
```

```
final_set
```

```
array([[1.0, 0.0, 0.0, 44.0, 72000.0],
       [0.0, 0.0, 1.0, 27.0, 48000.0],
       [0.0, 1.0, 0.0, 30.0, 54000.0],
       [0.0, 0.0, 1.0, 38.0, 61000.0],
       [0.0, 1.0, 0.0, 40.0, 63777.7777777778],
       [1.0, 0.0, 0.0, 35.0, 58000.0],
       [0.0, 0.0, 1.0, 38.77777777777778, 52000.0],
       [1.0, 0.0, 0.0, 48.0, 79000.0],
       [0.0, 1.0, 0.0, 50.0, 83000.0],
       [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc=StandardScaler()
```

```
sc.fit(final_set)
```

```
feat_standard_scaler=sc.transform(final_set)
```

```
feat_standard_scaler
```

```
→ array([[ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
          7.58874362e-01,  7.49473254e-01],
         [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
          -1.71150388e+00, -1.43817841e+00],
         [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
          -1.27555478e+00, -8.91265492e-01],
         [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
          -1.13023841e-01, -2.53200424e-01],
         [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
          1.77608893e-01,  6.63219199e-16],
         [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
          -5.48972942e-01, -5.26656882e-01],
         [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
          0.00000000e+00, -1.07356980e+00],
         [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
          1.34013983e+00,  1.38753832e+00],
         [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
          1.63077256e+00,  1.75214693e+00],
         [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
          -2.58340208e-01,  2.93712492e-01]])
```

```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler(feature_range=(0,1))
mms.fit(final_set)
feat_minmax_scaler=mms.transform(final_set)
feat_minmax_scaler
```

```
→ array([[1.        , 0.        , 0.        , 0.73913043, 0.68571429],
         [0.        , 0.        , 1.        , 0.        , 0.        ],
         [0.        , 1.        , 0.        , 0.13043478, 0.17142857],
         [0.        , 0.        , 1.        , 0.47826087, 0.37142857],
         [0.        , 1.        , 0.        , 0.56521739, 0.45079365],
         [1.        , 0.        , 0.        , 0.34782609, 0.28571429],
         [0.        , 0.        , 1.        , 0.51207729, 0.11428571],
         [1.        , 0.        , 0.        , 0.91304348, 0.88571429],
         [0.        , 1.        , 0.        , 1.        , 1.        ],
         [1.        , 0.        , 0.        , 0.43478261, 0.54285714]])
```

11.LINEAR REGRESSION

NAME : HARSHA VARDHANAN V

CLASS: CSE-B

ROLL NO : 230701108

```
import numpy as np import pandas as pd  
df=pd.read_csv('Salary_data.csv') df.info()
```

```
→ <class 'pandas.core.frame.DataFrame'>  
RangeIndex: 30 entries, 0 to 29  
Data columns (total 2 columns):  
 #   Column           Non-Null Count  Dtype     
---  --     
 0   YearsExperience 30 non-null      float64  
 1   Salary           30 non-null      float64  
dtypes: float64(2)  
memory usage: 608.0 bytes
```

```
df.dropna(inplace=True)
```

```
df.info()
```

```
→ <class 'pandas.core.frame.DataFrame'>  
RangeIndex: 30 entries, 0 to 29  
Data columns (total 2 columns):  
 #   Column           Non-Null Count  Dtype     
---  --     
 0   YearsExperience 30 non-null      float64  
 1   Salary           30 non-null      float64  
dtypes: float64(2)  
memory usage: 608.0 bytes
```

```
df.describe()
```

The screenshot shows a Jupyter Notebook cell with a dark theme. It displays a table of summary statistics for two columns: 'YearsExperience' and 'Salary'. The table includes rows for count, mean, std, min, 25%, 50%, 75%, and max. The 'Salary' column has values formatted with commas and scientific notation.

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

```

features=df.iloc[:,[0]].values
label=df.iloc[:,[1]].values
from sklearn.model_selection import train_test_split

# Assuming `features` and `label` are already defined in your code
x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.2, random_state=42)

```

```

from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)

```

The screenshot shows a Jupyter Notebook cell with a dark theme. A callout box highlights the creation of a 'LinearRegression()' object, which is highlighted in red.

```
model.score(x_train,y_train)
```

The screenshot shows a Jupyter Notebook cell with a dark theme. The output cell displays the score of the training set as 0.9645401573418146.

```
model.score(x_test,y_test)
```

The screenshot shows a Jupyter Notebook cell with a dark theme. The output cell displays the score of the test set as 0.9024461774180497.

```
model.coef_
```

```
→ array([[9423.81532303]])
```

```
import pickle
```

```
pickle.dump(model,open('SalaryPred.model','wb'))
```

```
model=pickle.load(open('SalaryPred.model','rb'))
```

```
yr_of_exp=float(input("Enter Years of Experience: "))
```

```
yr_of_exp_NP=np.array([[yr_of_exp]])
```

```
Salary=model.predict(yr_of_exp_NP)
```

```
→ Enter Years of Experience: 44
```

```
print("Estimated Salary for {} years of experience is {}: ".format(yr_of_exp,Salary))
```

```
→ Estimated Salary for 44.0 years of experience is [[439969.45722514]]:
```

12. LOGISTIC REGRESSION

NAME : HARSHA VARDHANAN V

ROLL NO : 230701108

```
import numpy as np import pandas as pd  
df=pd.read_csv('Social_Network_Ads.csv') df
```

	User ID	Gender	Age	EstimatedSalary	Purchased	
0	15624510	Male	19	19000	0	
1	15810944	Male	35	20000	0	
2	15668575	Female	26	43000	0	
3	15603246	Female	27	57000	0	
4	15804002	Male	19	76000	0	
...	
395	15691863	Female	46	41000	1	
396	15706071	Male	51	23000	1	
397	15654296	Female	50	20000	1	
398	15755018	Male	36	33000	0	
399	15594041	Female	49	36000	1	

400 rows × 5 columns

```
df.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased	
0	15624510	Male	19	19000	0	
1	15810944	Male	35	20000	0	
2	15668575	Female	26	43000	0	
3	15603246	Female	27	57000	0	
4	15804002	Male	19	76000	0	

```
features=df.iloc[:,[2,3]].values
```

```
label=df.iloc[:,4].values
```

features

```

Σ [ 38, 65000],  

[ 47, 51000],  

[ 47, 105000],  

[ 41, 63000],  

[ 53, 72000],  

[ 54, 108000],  

[ 39, 77000],  

[ 38, 61000],  

[ 38, 113000],  

[ 37, 75000],  

[ 42, 90000],  

[ 37, 57000],  

[ 36, 99000],  

[ 60, 34000],  

[ 54, 70000],  

[ 41, 72000],  

[ 40, 71000],  

[ 42, 54000],  

[ 43, 129000],  

[ 53, 34000],  

[ 47, 50000],  

[ 42, 79000],  

[ 42, 104000],  

[ 59, 29000],  

[ 58, 47000],  

[ 46, 88000],  

[ 38, 71000],  

[ 54, 26000],  

[ 60, 46000],  

[ 60, 83000],

```

label

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
```

```
for i in range(1, 401):
```

```
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.2, random_state=i)

# Initialize the Logistic Regression model
model = LogisticRegression()

# Train the model
model.fit(x_train, y_train)

# Calculate the train and test scores
train_score = model.score(x_train, y_train)
test_score = model.score(x_test, y_test)

# Print if test score is greater than train score
if test_score > train_score:
    print("Test {} Train {} Random State {}".format(test_score, train_score, i))
```

```
Test 0.8625 Train 0.8375 Random State 268
Test 0.875 Train 0.840625 Random State 275
Test 0.8625 Train 0.85 Random State 276
Test 0.925 Train 0.8375 Random State 277
Test 0.875 Train 0.846875 Random State 282
Test 0.85 Train 0.846875 Random State 283
Test 0.85 Train 0.84375 Random State 285
Test 0.9125 Train 0.834375 Random State 286
Test 0.85 Train 0.840625 Random State 290
Test 0.85 Train 0.840625 Random State 291
Test 0.85 Train 0.846875 Random State 292
Test 0.8625 Train 0.8375 Random State 294
Test 0.8875 Train 0.828125 Random State 297
Test 0.8625 Train 0.834375 Random State 300
Test 0.8625 Train 0.85 Random State 301
Test 0.8875 Train 0.85 Random State 302
Test 0.875 Train 0.846875 Random State 303
Test 0.8625 Train 0.834375 Random State 305
Test 0.9125 Train 0.8375 Random State 306
Test 0.875 Train 0.846875 Random State 308
Test 0.9 Train 0.84375 Random State 311
Test 0.8625 Train 0.834375 Random State 313
Test 0.9125 Train 0.834375 Random State 314
Test 0.875 Train 0.8375 Random State 315
Test 0.9 Train 0.846875 Random State 317
Test 0.9125 Train 0.821875 Random State 319
Test 0.8625 Train 0.85 Random State 321
Test 0.9125 Train 0.828125 Random State 322
Test 0.85 Train 0.846875 Random State 328
Test 0.85 Train 0.8375 Random State 332
Test 0.8875 Train 0.853125 Random State 336
Test 0.85 Train 0.8375 Random State 337
```

✓ 0s completed at 1:57PM

```
# Assuming features and label are defined earlier in your code
```

```
x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.2)
```

```
finalModel = LogisticRegression()
finalModel.fit(x_train, y_train)
```



LogisticRegression ⓘ ⓘ

LogisticRegression()

```
print(finalModel.score(x_train,y_train))
```

```
print(finalModel.score(x_test,y_test))
```

```
print(finalModel.score(x_test,y_test))  
→ 0.859375  
0.8375
```

```
from sklearn.metrics import classification_report  
print(classification_report(label,finalModel.predict(features)))
```

	precision	recall	f1-score	support
0	0.86	0.92	0.89	257
1	0.84	0.73	0.78	143
accuracy			0.85	400
macro avg	0.85	0.83	0.84	400
weighted avg	0.85	0.85	0.85	400